Authors: Hofmarcher

Date: 20-03-2023

This file is part of the "Deep Reinforcement Learning" lecture material. The following copyright statement applies to all code within this file.

Enable GPU Acceleration

Before you start exploring this notebook make sure that GPU support is enabled. To enable the GPU backend for your notebook, go to **Edit → Notebook Settings** and set **Hardware accelerator** to **GPU**.

Imports

Install Gymnasium and dependencies to render the environments

```
!apt-get update
!apt-get install -y swig python3-numpy python3-dev cmake zlib1g-dev libjpeg-dev xvfb ffmpeg xorg-dev python3-opengl libboost
!pip install gymnasium==0.29.0 gymnasium[box2d] pyvirtualdisplay imageio-ffmpeg moviepy==1.0.3
!pip install onnx onnx2pytorch==0.4.1
```

```
    Attempting uninstall: nvidia-cusparse-cu12
      Found existing installation: nvidia-cusparse-cu12 12.5.1.3
      Uninstalling nvidia-cusparse-cu12-12.5.1.3:
        Successfully uninstalled nvidia-cusparse-cu12-12.5.1.3
    Attempting uninstall: nvidia-cudnn-cu12
      Found existing installation: nvidia-cudnn-cu12 9.3.0.75
      Uninstalling nvidia-cudnn-cu12-9.3.0.75:
        Successfully uninstalled nvidia-cudnn-cu12-9.3.0.75
    Attempting uninstall: nvidia-cusolver-cu12
      Found existing installation: nvidia-cusolver-cu12 11.6.3.83
      Uninstalling nvidia-cusolver-cu12-11.6.3.83:
        Successfully uninstalled nvidia-cusolver-cu12-11.6.3.83
  Successfully installed nvidia-cublas-cu12-12.4.5.8 nvidia-cuda-cupti-cu12-12.4.127 nvidia-cuda-nvrtc-cu12-12.4.127 nvidia-cuda-run
```

```
%matplotlib inline

# Auxiliary Python imports
import os
import math
import io
import base64
import random
import shutil
from time import time, strftime
from glob import glob
from tqdm import tqdm
import numpy as np

# Pytorch
import torch
import torch.nn as nn
from torch.distributions.categorical import Categorical
import onnx
from onnx2pytorch import ConvertModel

# Environment import and set logger level to display error only
import gymnasium as gym
from gymnasium.spaces import Box
from gymnasium import logger as gymlogger
from gymnasium.wrappers import RecordVideo
gymlogger.set_level(gym.logger.ERROR)

# Plotting and notebook imports
import matplotlib.pyplot as plt
from matplotlib import animation
from IPython.display import HTML, clear_output
from IPython import display
```

Select device for training

By default we train on GPU if one is available, otherwise we fall back to the CPU. If you want to always use the CPU change accordingly.

```
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Device: " + str(device))
```

```
Device: cuda
```

Setup Google Drive mount to store your results

```
use_google_drive = True
if use_google_drive:
    from google.colab import drive
    drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Download Dataset and Expert model

```
# Download training and validation datasets
!wget --no-check-certificate 'https://cloud.ml.jku.at/s/citYJKPgmAGrHGy/download' -O expert.onnx
!wget --no-check-certificate 'https://cloud.ml.jku.at/s/yJ2ZsfqTos3Jn9y/download' -O train.zip
!wget --no-check-certificate 'https://cloud.ml.jku.at/s/3DxHLiqxTddepp8/download' -O val.zip

# Unzip datasets
!unzip -q -o train.zip
!unzip -q -o val.zip
```

```
--2025-03-25 20:59:53-- https://cloud.ml.jku.at/s/citYJKPgmAGrHGy/download
Resolving cloud.ml.jku.at (cloud.ml.jku.at)... 140.78.90.41
Connecting to cloud.ml.jku.at (cloud.ml.jku.at)|140.78.90.41|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 6747975 (6.4M) [application/octet-stream]
Saving to: 'expert.onnx'

expert.onnx         100%[===================>]   6.43M   947KB/s    in 7.2s

2025-03-25 21:00:02 (911 KB/s) - 'expert.onnx' saved [6747975/6747975]

--2025-03-25 21:00:02-- https://cloud.ml.jku.at/s/yJ2ZsfqTos3Jn9y/download
Resolving cloud.ml.jku.at (cloud.ml.jku.at)... 140.78.90.41
Connecting to cloud.ml.jku.at (cloud.ml.jku.at)|140.78.90.41|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 85373838 (81M) [application/zip]
Saving to: 'train.zip'

train.zip           100%[===================>]  81.42M  1.58MB/s    in 41s

2025-03-25 21:00:44 (1.96 MB/s) - 'train.zip' saved [85373838/85373838]

--2025-03-25 21:00:44-- https://cloud.ml.jku.at/s/3DxHLiqxTddepp8/download
Resolving cloud.ml.jku.at (cloud.ml.jku.at)... 140.78.90.41
Connecting to cloud.ml.jku.at (cloud.ml.jku.at)|140.78.90.41|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 16290406 (16M) [application/zip]
Saving to: 'val.zip'

val.zip             100%[===================>]  15.54M  7.73MB/s    in 2.0s

2025-03-25 21:00:47 (7.73 MB/s) - 'val.zip' saved [16290406/16290406]
```

```
import numpy as np
import os

def convert_to_4stack(in_dir="train", out_dir="train_4stack"):
        # 1) 读取所有 npz 文件
        files = sorted(glob(os.path.join(in_dir, "*.npz")))
        if not files:
                print(f"No npz files found in {in_dir}!")
                return

        states = []
        actions = []
        for f in files:
                data = np.load(f)
                # 这里假设 data["state"] 形状是 (84,84)
                # 以及 data["action"] 是一个 int
                s = data["state"]
                a = data["action"]
                states.append(s)
                actions.append(a)

        # 2) 把单帧拼成 4帧 堆叠
        new_states = []
        new_actions = []

        # 从第3索引开始，才能拿到 [i-3, i-2, i-1, i]
        # 所以有效 i ∈ [3, len(states)-1]
        for i in range(3, len(states)):
                # 堆叠 states[i-3], states[i-2], states[i-1], states[i] 在 axis=0 上
                stack_4 = np.stack(states[i-3:i+1], axis=0)   # shape (4,84,84)
```

```
            new_states.append(stack_4)
            new_actions.append(actions[i])   # 用最后一帧 i 的动作

    # 3) 把新数据写到 out_dir
    os.makedirs(out_dir, exist_ok=True)
    for i in range(len(new_states)):
        # 给新文件起个类似 000000.npz 的名字
        out_file = os.path.join(out_dir, f"{i:06d}.npz")
        np.savez_compressed(out_file, state=new_states[i], action=new_actions[i])

    print(f"Converted {len(new_states)} samples and saved to {out_dir}/")

# 以 train 为例
convert_to_4stack(in_dir="train", out_dir="train_4stack")
# val 同理
convert_to_4stack(in_dir="val", out_dir="val_4stack")
```

```
Converted 49535 samples and saved to train_4stack/
Converted 9455 samples and saved to val_4stack/
```

```
train_files = [f for f in os.listdir('train') if f.endswith('.npz')]
print("Train NPZ files:", train_files)

# 随便拿一个 npz 文件试试看
example_file = os.path.join('train', train_files[0])
data = np.load(example_file)
print("Keys in this NPZ:", list(data.keys()))

# 每个 key 通常对应一个 numpy 数组，查看形状:
for k in data.keys():
    print(k, data[k].shape)
```

```
Train NPZ files: ['023843.npz', '003214.npz', '012092.npz', '017992.npz', '032313.npz', '032514.npz', '038229.npz', '010959.npz', '0
Keys in this NPZ: ['state', 'action']
state (84, 84)
action ()
```

```
train_files = [f for f in os.listdir('train_4stack') if f.endswith('.npz')]
print("Train NPZ files:", train_files)

# 随便拿一个 npz 文件试试看
example_file = os.path.join('train_4stack', train_files[0])
data = np.load(example_file)
print("Keys in this NPZ:", list(data.keys()))

# 每个 key 通常对应一个 numpy 数组，查看形状:
for k in data.keys():
    print(k, data[k].shape)
```

```
Train NPZ files: ['023843.npz', '003214.npz', '012092.npz', '017992.npz', '032313.npz', '032514.npz', '038229.npz', '010959.npz', '0
Keys in this NPZ: ['state', 'action']
state (4, 84, 84)
action ()
```

```
import onnx

# 假设你的文件名是 'expert.onnx'
model_path = "expert.onnx"

# 加载模型
model = onnx.load(model_path)

# 检查模型是否格式正确 (可选)
onnx.checker.check_model(model)

# 可以打印模型的大致结构
print(onnx.helper.printable_graph(model.graph))
```

```
graph torch-jit-export (
  %input.1[FLOAT, 1x4x84x84]
) initializers (
  %actor.bias[FLOAT, 5]
  %actor.weight[FLOAT, 5x512]
  %network.0.bias[FLOAT, 32]
  %network.0.weight[FLOAT, 32x4x8x8]
  %network.2.bias[FLOAT, 64]
  %network.2.weight[FLOAT, 64x32x4x4]
  %network.4.bias[FLOAT, 64]
  %network.4.weight[FLOAT, 64x64x3x3]
  %network.7.bias[FLOAT, 512]
  %network.7.weight[FLOAT, 512x3136]
) {
  %11 = Conv[dilations = [1, 1], group = 1, kernel_shape = [8, 8], pads = [0, 0, 0, 0], strides = [4, 4]](%input.1, %network.0.weigh
  %12 = Relu(%11)
  %13 = Conv[dilations = [1, 1], group = 1, kernel_shape = [4, 4], pads = [0, 0, 0, 0], strides = [2, 2]](%12, %network.2.weight, %n
  %14 = Relu(%13)
  %15 = Conv[dilations = [1, 1], group = 1, kernel_shape = [3, 3], pads = [0, 0, 0, 0], strides = [1, 1]](%14, %network.4.weight, %n
  %16 = Relu(%15)
  %17 = Flatten[axis = 1](%16)
  %18 = Gemm[alpha = 1, beta = 1, transB = 1](%17, %network.7.weight, %network.7.bias)
  %19 = Relu(%18)
  %20 = Gemm[alpha = 1, beta = 1, transB = 1](%19, %actor.weight, %actor.bias)
  return %20
}
```

```python
def get_tensor_shape(value_info):
    """解析 ONNX 的 tensor shape，并返回 Python 列表。"""
    shape = []
    tensor_type = value_info.type.tensor_type
    for dim in tensor_type.shape.dim:
        # 如果 dim_value 为 0 或 -1，可能表示动态维度
        if dim.dim_value > 0:
            shape.append(dim.dim_value)
        else:
            # 某些模型使用动态维度，可以用 None 或 "?" 标识
            shape.append("?")
    return shape

# 查看模型的输入形状
print("== Model Inputs ==")
for i, input_tensor in enumerate(model.graph.input):
    shape = get_tensor_shape(input_tensor)
    print(f"Input {i} name:", input_tensor.name)
    print(f"Input {i} shape:", shape)

# 查看模型的输出形状
print("\n== Model Outputs ==")
for i, output_tensor in enumerate(model.graph.output):
    shape = get_tensor_shape(output_tensor)
    print(f"Output {i} name:", output_tensor.name)
    print(f"Output {i} shape:", shape)
```

```
== Model Inputs ==
Input 0 name: input.1
Input 0 shape: [1, 4, 84, 84]

== Model Outputs ==
Output 0 name: 20
Output 0 shape: [1, 5]
```

```python
import numpy as np

train_files = sorted(glob("train/*.npz"))
# 假设想看前3个文件
for f in train_files[:3]:
    data = np.load(f)
    state = data["state"]
    print(f"{f}: {state.shape}")
```

```
train/000001.npz: (84, 84)
train/000002.npz: (84, 84)
train/000003.npz: (84, 84)
```

```
import numpy as np

train_files = sorted(glob("train_4stack/*.npz"))
# 假设想看前3个文件
for f in train_files[:3]:
    data = np.load(f)
    state = data["state"]
    print(f"{f}: {state.shape}")
```

```
train_4stack/000000.npz: (4, 84, 84)
train_4stack/000001.npz: (4, 84, 84)
train_4stack/000002.npz: (4, 84, 84)
```

Auxiliary Methods

The following cell contains classes and functions to provide some functionality for logging, plotting and exporting your model in the format required by the submission server. You are free to use your own logging framework if you wish (such as tensorboard or Weights & Biases). The logger is a very simple implementation of a CSV-file based logger. Additionally it creates a folder for each run with subfolders for model files, logs and videos.

```
class Logger():
    def __init__(self, logdir, params=None):
        self.basepath = os.path.join(logdir, strftime("%Y-%m-%dT%H-%M-%S"))
        os.makedirs(self.basepath, exist_ok=True)
        os.makedirs(self.log_dir, exist_ok=True)
        if params is not None and os.path.exists(params):
            shutil.copyfile(params, os.path.join(self.basepath, "params.pkl"))
        self.log_dict = {}
        self.dump_idx = {}

    @property
    def param_file(self):
        return os.path.join(self.basepath, "params.pkl")

    @property
    def onnx_file(self):
        return os.path.join(self.basepath, "model.onnx")

    @property
    def video_dir(self):
        return os.path.join(self.basepath, "videos")

    @property
    def log_dir(self):
        return os.path.join(self.basepath, "logs")

    def log(self, name, value):
        if name not in self.log_dict:
            self.log_dict[name] = []
            self.dump_idx[name] = -1
        self.log_dict[name].append((len(self.log_dict[name]), time(), value))

    def get_values(self, name):
        if name in self.log_dict:
            return [x[2] for x in self.log_dict[name]]
        return None

    def dump(self):
        for name, rows in self.log_dict.items():
            with open(os.path.join(self.log_dir, name + ".log"), "a") as f:
                for i, row in enumerate(rows):
                    if i > self.dump_idx[name]:
                        f.write(",".join([str(x) for x in row]) + "\n")
                        self.dump_idx[name] = i


def plot_metrics(logger):
    train_loss  = logger.get_values("training_loss")
    train_entropy  = logger.get_values("training_entropy")
    val_loss = logger.get_values("validation_loss")
```

```
        val_acc = logger.get_values("validation_accuracy")

        fig = plt.figure(figsize=(15,5))
        ax1 = fig.add_subplot(131, label="train")
        ax2 = fig.add_subplot(131, label="val",frame_on=False)
        ax4 = fig.add_subplot(132, label="entropy")
        ax3 = fig.add_subplot(133, label="acc")

        ax1.plot(train_loss, color="C0")
        ax1.set_ylabel("Loss")
        ax1.set_xlabel("Update (Training)", color="C0")
        ax1.xaxis.grid(False)
        ax1.set_ylim((0,4))

        ax2.plot(val_loss, color="C1")
        ax2.xaxis.tick_top()
        ax2.yaxis.tick_right()
        ax2.set_xlabel('Epoch (Validation)', color="C1")
        ax2.xaxis.set_label_position('top')
        ax2.xaxis.grid(False)
        ax2.get_yaxis().set_visible(False)
        ax2.set_ylim((0,4))

        ax4.plot(train_entropy, color="C3")
        ax4.set_xlabel('Update (Training)', color="black")
        ax4.set_ylabel("Entropy", color="C3")
        ax4.tick_params(axis='x', colors="black")
        ax4.tick_params(axis='y', colors="black")
        ax4.xaxis.grid(False)

        ax3.plot(val_acc, color="C2")
        ax3.set_xlabel("Epoch (Validation)", color="black")
        ax3.set_ylabel("Accuracy", color="C2")
        ax3.tick_params(axis='x', colors="black")
        ax3.tick_params(axis='y', colors="black")
        ax3.xaxis.grid(False)
        ax3.set_ylim((0,1))

        fig.tight_layout(pad=2.0)
        plt.show()

"""
Utility functions to enable video recording of gym environment and displaying it
"""
def show_video(video_dir):
        mp4list = glob(f'{video_dir}/*.mp4')
        if len(mp4list) > 0:
                mp4 = mp4list[0]
                video = io.open(mp4, 'r+b').read()
                encoded = base64.b64encode(video)
                display.display(HTML(data='''<video alt="test" autoplay
                                        loop controls style="height: 400px;">
                                        <source src="data:video/mp4;base64,{0}" type="video/mp4" />
                                </video>'''.format(encoded.decode('ascii'))))
        else:
                print("Could not find video")

def save_as_onnx(torch_model, sample_input, model_path):
        torch.onnx.export(torch_model,                          # model being run
                                        sample_input,                          # model input (or a tuple for multiple inputs)
                                        f=model_path,                          # where to save the model (can be a file or f
                                        export_params=True,                # store the trained parameter weights inside the model
                                        opset_version=17,                    # the ONNX version to export the model to - see h
                                        do_constant_folding=True,    # whether to execute constant folding for optimization
                                        )
```

Dataset

\Use this dataset class to load the provided demonstrations. Furthermore, this dataset has functionality to add new samples to the dataset which you will need for implementing the DAgger algorithm.

```
class DemonstrationDataset(torch.utils.data.Dataset):
    def __init__(self, data_dir):
        self.data_dir = data_dir
        self.files = sorted(glob(f"{data_dir}/*.npz"))

    def __len__(self):
        return len(self.files)

    def __getitem__(self, idx):
        data = np.load(self.files[idx])
        #state = data["state"][np.newaxis, ...].astype(np.float32)
        state = data["state"].astype(np.float32)
        action = data["action"]
        return state / 255.0, action.item()

    def append(self, states, actions):
        offset = len(self) + 1
        for i in range(len(states)):
            filename = f"{self.data_dir}/{offset+i:06}.npz"
            np.savez_compressed(filename, state=states[i], action=actions[i].astype(np.int32))
            self.files.append(filename)
```

Inspect data

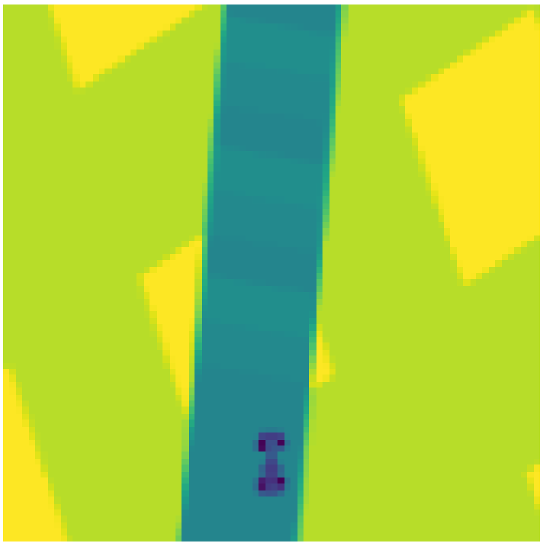
It is always a good idea to take a look at the data when you start working with a new dataset. Feel free to investigate the dataset further on your own.

```
# Action Statistics
dataset = DemonstrationDataset("train_4stack")
print("Number of samples: {}".format(len(dataset)));
```

```
Number of samples: 49535
```

```
# Action mapping from gymnasium.farama.org
action_mapping = {
    0: "do nothing",
    1: "steer left",
    2: "steer right",
    3: "gas",
    4: "brake"
}

# Visualize random frames
idx = np.random.randint(len(dataset))
state, action = dataset[idx]
# store a single frame as we need it later for exporting an ONNX model (it needs a sample of the input for the e›
sample_state = torch.Tensor(state).unsqueeze(0).to(device)
# Display the sample
print(f"Action: {action_mapping[action]}")
plt.axis("off")
plt.imshow(state[0]);
```

```
Action: gas
```



```
# release memory
del dataset
```

Define Policy Network

You need to design a neural network architecture that is capable of mapping a state to an action. The input is a single image with the following properties:

- Resolution of 84x84 pixels
- Grayscale (meaning a single channel as opposed to three channels of an RGB image)
- The values of each pixel should be between 0 and 1

The output of the network should be one unit per possible action, as our environment has 5 actions that results in 5 output units. Your network must implement the forward function in order to be compatible with the evaluation script.

```
import torch.nn.functional as F

class PolicyNetwork(nn.Module):
    def __init__(self, n_units_out, dropout_p=0.2):
        """
        n_units_out: 动作数 (5)
        dropout_p: 全连接层的dropout概率，简单正则化
        """
        super(PolicyNetwork, self).__init__()

        # 注意：输入是(4,84,84)，因为FrameStack=4帧叠加（灰度）
        # 如果只是一帧，可将 in_channels 调成1。题目中要求和专家一样堆叠了4帧，则 in_channels=4。
        self.conv1 = nn.Conv2d(in_channels=4, out_channels=32, kernel_size=8, stride=4)
        self.conv2 = nn.Conv2d(in_channels=32, out_channels=64, kernel_size=4, stride=2)
        self.conv3 = nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3, stride=1)

        self.relu = nn.ReLU()
        self.flatten = nn.Flatten()

        # 理论上经过上述卷积后尺寸: (4,84,84) -> (32, 20,20) -> (64,9,9) -> (64,7,7)
        # 最终扁平化是 64*7*7=3136，这里和expert.onnx匹配
        self.fc1 = nn.Linear(64*7*7, 512)
        self.dropout = nn.Dropout(dropout_p)
        self.fc2 = nn.Linear(512, n_units_out)   # 输出5个动作的logits

    def forward(self, x):
        """
        x shape: (B, 4, 84, 84)
        return shape: (B, 5)
        """
        x = self.relu(self.conv1(x))
        x = self.relu(self.conv2(x))
        x = self.relu(self.conv3(x))
```

```
        x = self.flatten(x)
        x = self.relu(self.fc1(x))
        x = self.dropout(x)
        x = self.fc2(x)
        return x


import torch
import torch.nn as nn
import torch.nn.functional as F

class PolicyNetwork(nn.Module):

    def __init__(self, n_units_out, dropout_p=0.2, num_groups=4):
        """
        n_units_out: 动作数 (5)
        dropout_p: 全连接层的dropout概率，简单正则化
        num_groups: GroupNorm 的分组数；需要整除通道数
        """
        super(PolicyNetwork, self).__init__()

        # 卷积部分：3 层卷积 + GroupNorm + ReLU
        # 注意 out_channels 需要被 num_groups 整除
        self.conv1 = nn.Conv2d(in_channels=4, out_channels=32, kernel_size=8, stride=4)
        self.gn1   = nn.GroupNorm(num_groups=num_groups, num_channels=32)

        self.conv2 = nn.Conv2d(in_channels=32, out_channels=64, kernel_size=4, stride=2)
        self.gn2   = nn.GroupNorm(num_groups=num_groups, num_channels=64)

        self.conv3 = nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3, stride=1)
        self.gn3   = nn.GroupNorm(num_groups=num_groups, num_channels=64)

        # 展平
        self.flatten = nn.Flatten()

        # 全连接部分：一层 Linear + LayerNorm + ReLU + Dropout，最后再输出 Linear
        self.fc1  = nn.Linear(64 * 7 * 7, 512)
        self.ln1  = nn.LayerNorm(512)
        self.drop = nn.Dropout(dropout_p)
        self.fc2  = nn.Linear(512, n_units_out)

    def forward(self, x):
        """
        x shape: (batch_size, 4, 84, 84)
        return shape: (batch_size, 5)
        """
        # 卷积 + GroupNorm + ReLU
        x = F.relu(self.gn1(self.conv1(x)))
        x = F.relu(self.gn2(self.conv2(x)))
        x = F.relu(self.gn3(self.conv3(x)))

        x = self.flatten(x)          # shape => (batch_size, 64*7*7 = 3136)
        x = self.fc1(x)                  # => (batch_size, 512)
        x = self.ln1(x)                  # LayerNorm
        x = F.relu(x)
        x = self.drop(x)
        x = self.fc2(x)                  # => (batch_size, 5)
        return x
```

Train behavioral cloning policy

Now that you have a Dataset and a network you need to train your network. With behavioral cloning we want to imitate the behavior of the agent that produced the demonstration dataset as close as possible. This is basically supervised learning, where you want to minimize the loss of your network on the training and validation sets.

Some tips as to what you need to implement:

- choose the appropriate loss function (think on which kind of problem you are solving)
- choose an optimizer and its hyper-parameters
- optional: use a learning-rate scheduler
- don't forget to evaluate your network on the validation set

- store your model and training progress often so you don't loose progress if your program crashes

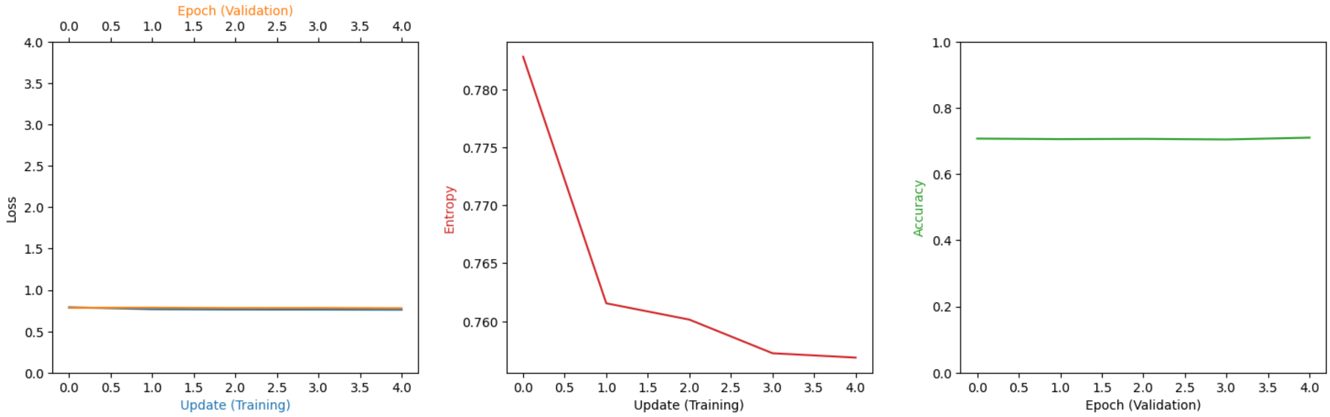
In case you use the provided Logger:

- `logger.log("training_loss", <loss-value>)` to log a particular value
- `logger.dump()` to write the current logs to a log file (e.g. after every episode)
- `logger.log_dir`, `logger.param_file`, `logger.onnx_file`, `logger.video_dir` point to files or directories you can use to save files
- you might want to specify your google drive folder as a logdir in order to automatically sync your results
- if you log the metrics specified in the `plot_metrics` function you can use it to visualize your training progress (or take it as a template to plot your own metrics)

```
import torch.optim as optim

# choose the batchsize for training
batch_size = 64

# Datasets
train_set = DemonstrationDataset("train_4stack")
train_loader = torch.utils.data.DataLoader(train_set, batch_size=batch_size, num_workers=2, shuffle=True, drop_last=False, pin_mer
val_set = DemonstrationDataset("val_4stack")
val_loader = torch.utils.data.DataLoader(val_set, batch_size=batch_size, num_workers=2, shuffle=False, drop_last=False, pin_memory

# Specify the google drive mount here if you want to store logs and weights there (and set it up earlier)
# You can also choose to use a different logging framework such as tensorboard (not recommended on Colab) or Weights
logger = Logger("logdir")
print("Saving state to {}".format(logger.basepath))

# Network
model = PolicyNetwork(n_units_out=5)
model = model.to(device)
num_trainable_params = sum(p.numel() for p in model.parameters() if p.requires_grad)
print("Trainable Parameters: {}".format(num_trainable_params))

######################
### YOUR CODE HERE ###
######################
# Implement your training and evaluation loop
# feel free to define your own functions for training and evaluation

# If you want to export your model as an ONNX file use the following code as template
# If you use the provided logger you can use this directly

optimizer = optim.Adam(model.parameters(), lr=1e-4, weight_decay=1e-5)
criterion = nn.CrossEntropyLoss()

def train_one_epoch(model, dataloader, optimizer, logger=None):
        model.train()
        total_loss = 0.0
        total_entropy = 0.0
        total_samples = 0

        for states, actions in dataloader:
                # states shape: (B, 1, 84, 84),  但在环境中是4帧stack，因此这里需要扩维
                #   如果已经在 DemonstrationDataset 做了 (1,84,84)，则仍需注意输入通道维度
                #   这里假设示例数据中实际只有单帧，那么可以简单 replicate 到4通道
                #   或者你在收集数据时就是4帧堆叠，这要看你数据本身如何准备。
                # 如果演示数据只有单帧，可用 repeat 在通道维度扩成4。
                # 如果你已经保证数据本身就是(4,84,84)，就不需要这一步了。
                states = states.to(device)   # shape (B, 1, 84, 84)
                #states = states.repeat(1,4,1,1)   # (B,4,84,84)——仅当原始数据只有1帧时

                actions = actions.to(device)

                logits = model(states)
                loss = criterion(logits, actions)

                # 计算分类分布的熵（仅作monitor）
                dist = Categorical(logits=logits)
                entropy = dist.entropy().mean()

                optimizer.zero_grad()
                loss.backward()
                optimizer.step()
```

```
            # 累计
            bs = states.size(0)
            total_loss += loss.item() * bs
            total_entropy += entropy.item() * bs
            total_samples += bs

    avg_loss = total_loss / total_samples
    avg_entropy = total_entropy / total_samples

    if logger is not None:
        logger.log("training_loss", avg_loss)
        logger.log("training_entropy", avg_entropy)
        logger.dump()

    return avg_loss, avg_entropy


def validate(model, dataloader, logger=None):
    model.eval()
    total_loss = 0.0
    total_correct = 0
    total_samples = 0

    with torch.no_grad():
        for states, actions in dataloader:
            states = states.to(device)
            # 同理，如需把单帧数据扩成4帧
            #states = states.repeat(1,4,1,1)

            actions = actions.to(device)

            logits = model(states)
            loss = criterion(logits, actions)

            # 计算accuracy
            preds = logits.argmax(dim=1)
            correct = (preds == actions).sum().item()

            bs = states.size(0)
            total_loss += loss.item() * bs
            total_correct += correct
            total_samples += bs

    avg_loss = total_loss / total_samples
    avg_acc = total_correct / total_samples

    if logger is not None:
        logger.log("validation_loss", avg_loss)
        logger.log("validation_accuracy", avg_acc)
        logger.dump()

    return avg_loss, avg_acc


# ---------------------------
# 训练多个epoch
# ---------------------------
n_epochs = 5
for epoch in range(n_epochs):
    print(f"Epoch [{epoch+1}/{n_epochs}]")
    train_loss, train_entropy = train_one_epoch(model, train_loader, optimizer, logger=logger)
    val_loss, val_acc = validate(model, val_loader, logger=logger)
    print(f"  Train Loss: {train_loss:.4f}, Entropy: {train_entropy:.4f} | Val Loss: {val_loss:.4f}, Val Acc: {val_acc:.

plot_metrics(logger)


save_as_onnx(model, sample_state, logger.onnx_file)
```

```
Saving state to logdir/2025-03-25T21-03-28
Trainable Parameters: 1688037
Epoch [1/5]
  Train Loss: 0.7903, Entropy: 0.7828 | Val Loss: 0.7832, Val Acc: 0.7072
Epoch [2/5]
  Train Loss: 0.7664, Entropy: 0.7615 | Val Loss: 0.7848, Val Acc: 0.7057
Epoch [3/5]
  Train Loss: 0.7623, Entropy: 0.7601 | Val Loss: 0.7806, Val Acc: 0.7064
Epoch [4/5]
  Train Loss: 0.7618, Entropy: 0.7572 | Val Loss: 0.7814, Val Acc: 0.7046
Epoch [5/5]
  Train Loss: 0.7602, Entropy: 0.7569 | Val Loss: 0.7786, Val Acc: 0.7101
```



Evaluate the agent in the real environment

Environment and Agent

We provide some wrappers you need in order to get the same states from the environment as in the demonstration dataset. Additionally the `RecordState` wrapper should be very helpful in collecting new samples for the DAgger algorithm.

```
class CropObservation(gym.ObservationWrapper):
    def __init__(self, env, shape):
        gym.ObservationWrapper.__init__(self, env)
        self.shape = shape
        obs_shape = self.shape + env.observation_space.shape[2:]
        self.observation_space = Box(low=0, high=255, shape=obs_shape, dtype=np.uint8)

    def observation(self, observation):
        return observation[:self.shape[0], :self.shape[1]]


class RecordState(gym.Wrapper):
    def __init__(self, env: gym.Env, reset_clean: bool = True):
        gym.Wrapper.__init__(self, env)

        assert env.render_mode is not None
        self.frame_list = []
        self.reset_clean = reset_clean

    def step(self, action, **kwargs):
        output = self.env.step(action, **kwargs)
        self.frame_list.append(output[0])
        return output

    def reset(self, *args, **kwargs):
        result = self.env.reset(*args, **kwargs)

        if self.reset_clean:
            self.frame_list = []
        self.frame_list.append(result[0])

        return result

    def render(self):
```

```
            frames = self.frame_list
            self.frame_list = []
            return frames


class Agent():
    def __init__(self, model, device):
        self.model = model
        self.device = device

    def select_action(self, state):
        with torch.no_grad():
            state = torch.Tensor(state).unsqueeze(0).to(device) / 255.0 # rescale
            logits = self.model(state)
            if type(logits) is tuple:
                logits = logits[0]
            probs = Categorical(logits=logits)
            return probs.sample().cpu().numpy()[0]


def make_env(seed, capture_video=True):
    env = gym.make("CarRacing-v2", render_mode="rgb_array", continuous=False)
    env = gym.wrappers.RecordEpisodeStatistics(env)
    if capture_video:
        env = gym.wrappers.RecordVideo(env, logger.video_dir)

    env = CropObservation(env, (84, 96))
    env = gym.wrappers.ResizeObservation(env, (84, 84))
    env = gym.wrappers.GrayScaleObservation(env)
    env = RecordState(env, reset_clean=True)
    env = gym.wrappers.FrameStack(env, 4)
    env.reset(seed=seed)
    env.action_space.seed(seed)
    env.observation_space.seed(seed)
    return env


def run_episode(agent, show_progress=True, capture_video=True, seed=None):
    env = make_env(seed=seed, capture_video=capture_video)
    state, _ = env.reset()
    score = 0
    done = False
    if show_progress:
        progress = tqdm(desc="Score: 0")

    while not done:
        #action = agent.select_action(state[-1][np.newaxis, ...])
        action = agent.select_action(state)
        state, reward, terminated, truncated, _ = env.step(action)
        score += reward
        done = terminated or truncated
        if show_progress:
            progress.update()
            progress.set_description("Score: {:.2f}".format(score))
    env.close()

    if show_progress:
        progress.close()
    if capture_video:
        show_video(logger.video_dir)

    return score
```

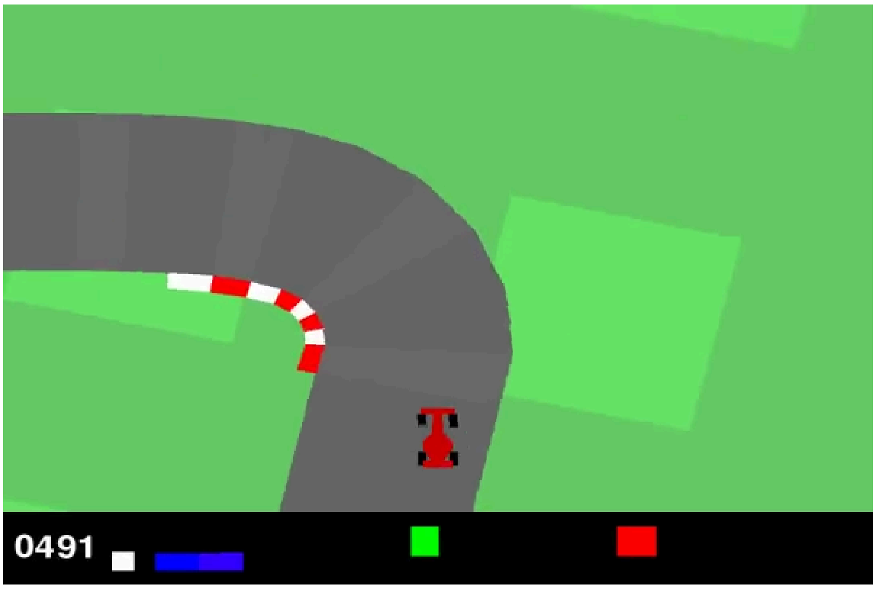
Evaluate behavioral cloning agent

Let's see how the agent is doing in the real environment

```
train_policy = Agent(model, device)
score = run_episode(train_policy, show_progress=True, capture_video=True);
print(f"Score: {score:.2f}")
```

```
Score: 0: 0it [00:00, ?it/s]<ipython-input-22-1ddebf4c0367>:47: UserWarning: Creating a tensor from a list of numpy.ndar
  state = torch.Tensor(state).unsqueeze(0).to(device) / 255.0 # rescale
Score: 718.00: : 999it [00:33, 36.21it/s]Moviepy - Building video /content/logdir/2025-03-25T21-03-28/videos/rl-video-ep
Moviepy - Writing video /content/logdir/2025-03-25T21-03-28/videos/rl-video-episode-0.mp4


t:   0%|          | 0/1001 [00:00<?, ?it/s, now=None]
t:   2%|▏         | 21/1001 [00:00<00:05, 182.74it/s, now=None]
t:   7%|▋         | 67/1001 [00:00<00:02, 334.66it/s, now=None]
t:  10%|█         | 102/1001 [00:00<00:05, 151.76it/s, now=None]
t:  12%|█▏        | 125/1001 [00:00<00:06, 138.91it/s, now=None]
t:  14%|█▍        | 144/1001 [00:00<00:06, 140.58it/s, now=None]
t:  16%|█▌        | 162/1001 [00:01<00:05, 143.59it/s, now=None]
t:  18%|█▊        | 179/1001 [00:01<00:05, 146.59it/s, now=None]
t:  20%|█▉        | 196/1001 [00:01<00:05, 148.24it/s, now=None]
t:  21%|██        | 212/1001 [00:01<00:05, 148.91it/s, now=None]
t:  23%|██▎       | 228/1001 [00:01<00:05, 150.64it/s, now=None]
t:  24%|██▍       | 244/1001 [00:01<00:05, 145.38it/s, now=None]
t:  26%|██▌       | 259/1001 [00:01<00:05, 138.21it/s, now=None]
t:  27%|██▋       | 274/1001 [00:01<00:05, 132.63it/s, now=None]
t:  29%|██▉       | 288/1001 [00:01<00:05, 128.40it/s, now=None]
t:  30%|███       | 301/1001 [00:02<00:05, 124.58it/s, now=None]
t:  31%|███▏      | 315/1001 [00:02<00:05, 125.88it/s, now=None]
t:  33%|███▎      | 328/1001 [00:02<00:05, 125.63it/s, now=None]
t:  34%|███▍      | 341/1001 [00:02<00:05, 121.66it/s, now=None]
t:  35%|███▌      | 354/1001 [00:02<00:05, 113.45it/s, now=None]
t:  37%|███▋      | 366/1001 [00:02<00:05, 114.32it/s, now=None]
t:  38%|███▊      | 378/1001 [00:02<00:05, 114.34it/s, now=None]
t:  39%|███▉      | 391/1001 [00:02<00:05, 118.45it/s, now=None]
t:  40%|████      | 403/1001 [00:02<00:05, 116.93it/s, now=None]
t:  41%|████▏     | 415/1001 [00:03<00:05, 114.85it/s, now=None]
t:  43%|████▎     | 430/1001 [00:03<00:04, 118.71it/s, now=None]
t:  44%|████▍     | 444/1001 [00:03<00:04, 123.94it/s, now=None]
t:  46%|████▌     | 457/1001 [00:03<00:04, 121.68it/s, now=None]
t:  47%|████▋     | 471/1001 [00:03<00:04, 124.22it/s, now=None]
t:  48%|████▊     | 484/1001 [00:03<00:04, 123.01it/s, now=None]
t:  50%|████▉     | 497/1001 [00:03<00:04, 116.02it/s, now=None]
t:  51%|█████     | 509/1001 [00:03<00:04, 114.20it/s, now=None]
t:  52%|█████▏    | 521/1001 [00:03<00:04, 114.32it/s, now=None]
t:  53%|█████▎    | 535/1001 [00:04<00:03, 118.24it/s, now=None]
t:  55%|█████▍    | 547/1001 [00:04<00:04, 108.31it/s, now=None]
t:  56%|█████▌    | 558/1001 [00:04<00:04, 102.55it/s, now=None]
t:  57%|█████▋    | 569/1001 [00:04<00:04, 103.88it/s, now=None]
t:  58%|█████▊    | 580/1001 [00:04<00:04, 97.41it/s, now=None]
t:  59%|█████▉    | 590/1001 [00:04<00:04, 91.92it/s, now=None]
t:  60%|██████    | 601/1001 [00:04<00:04, 92.55it/s, now=None]
t:  61%|██████    | 611/1001 [00:04<00:04, 87.51it/s, now=None]
t:  62%|██████▏   | 621/1001 [00:05<00:04, 85.18it/s, now=None]
t:  63%|██████▎   | 630/1001 [00:05<00:04, 83.94it/s, now=None]
t:  64%|██████▍   | 639/1001 [00:05<00:04, 78.48it/s, now=None]
t:  65%|██████▍   | 649/1001 [00:05<00:04, 80.96it/s, now=None]
t:  66%|██████▌   | 659/1001 [00:05<00:04, 84.30it/s, now=None]
t:  67%|██████▋   | 669/1001 [00:05<00:03, 88.19it/s, now=None]
t:  68%|██████▊   | 679/1001 [00:05<00:03, 90.36it/s, now=None]
t:  69%|██████▉   | 689/1001 [00:05<00:03, 88.68it/s, now=None]
t:  70%|██████▉   | 698/1001 [00:05<00:03, 85.99it/s, now=None]
t:  71%|███████   | 708/1001 [00:06<00:03, 88.20it/s, now=None]
t:  72%|███████▏  | 717/1001 [00:06<00:03, 88.46it/s, now=None]
t:  73%|███████▎  | 726/1001 [00:06<00:03, 88.38it/s, now=None]
t:  74%|███████▎  | 736/1001 [00:06<00:02, 90.79it/s, now=None]
t:  75%|███████▍  | 746/1001 [00:06<00:02, 92.45it/s, now=None]
t:  76%|███████▌  | 756/1001 [00:06<00:02, 89.86it/s, now=None]
t:  77%|███████▋  | 766/1001 [00:06<00:02, 86.38it/s, now=None]
t:  78%|███████▊  | 776/1001 [00:06<00:02, 90.09it/s, now=None]
t:  79%|███████▊  | 786/1001 [00:06<00:02, 89.19it/s, now=None]
t:  80%|███████▉  | 796/1001 [00:07<00:02, 90.19it/s, now=None]
t:  81%|████████  | 806/1001 [00:07<00:02, 90.40it/s, now=None]
t:  82%|████████▏ | 816/1001 [00:07<00:02, 87.60it/s, now=None]
t:  82%|████████▏ | 825/1001 [00:07<00:02, 86.22it/s, now=None]
t:  83%|████████▎ | 834/1001 [00:07<00:02, 82.76it/s, now=None]
t:  84%|████████▍ | 843/1001 [00:07<00:01, 82.78it/s, now=None]
t:  85%|████████▌ | 852/1001 [00:07<00:01, 77.02it/s, now=None]
t:  86%|████████▌ | 862/1001 [00:07<00:01, 75.90it/s, now=None]
t:  87%|████████▋ | 871/1001 [00:07<00:01, 78.85it/s, now=None]
t:  88%|████████▊ | 879/1001 [00:08<00:01, 72.20it/s, now=None]
t:  89%|████████▊ | 887/1001 [00:08<00:01, 71.89it/s, now=None]
t:  90%|████████▉ | 898/1001 [00:08<00:01, 81.64it/s, now=None]
t:  91%|█████████ | 907/1001 [00:08<00:01, 78.93it/s, now=None]
t:  92%|█████████▏| 916/1001 [00:08<00:01, 75.03it/s, now=None]
t:  92%|█████████▏| 924/1001 [00:08<00:01, 70.54it/s, now=None]
t:  93%|█████████▎| 932/1001 [00:08<00:00, 69.16it/s, now=None]
t:  94%|█████████▍| 939/1001 [00:08<00:00, 68.03it/s, now=None]
t:  95%|█████████▍| 947/1001 [00:09<00:00, 69.57it/s, now=None]
```

```
t:  95%|█████████ | 955/1001 [00:09<00:00, 68.90it/s, now=None]
t:  96%|█████████▋| 964/1001 [00:09<00:00, 73.87it/s, now=None]
t:  97%|█████████▋| 972/1001 [00:09<00:00, 72.89it/s, now=None]
t:  98%|█████████▊| 980/1001 [00:09<00:00, 68.94it/s, now=None]
t:  99%|█████████▉| 989/1001 [00:09<00:00, 69.43it/s, now=None]
t: 100%|█████████▉| 998/1001 [00:09<00:00, 69.51it/s, now=None]
Score: 717.90: : 1000it [00:43, 22.74it/s]Moviepy - Done !
Moviepy - video ready /content/logdir/2025-03-25T21-03-28/videos/rl-video-episode-0.mp4
```




```
Score: 717.90
```

Since we often have high variance when evaluating RL agents we should evaluate the agent multiple times to get a better feeling for its performance.

```
train_policy = Agent(model, device)
n_eval_episodes = 10
scores = []
for i in tqdm(range(n_eval_episodes), desc="Episode"):
        scores.append(run_episode(train_policy, show_progress=False, capture_video=False))
        print("Score: %d" % scores[-1])
print("Mean Score: %.2f (Std: %.2f)" %(np.mean(scores), np.std(scores)))
```

```
Episode:  10%|█         | 1/10 [00:17<02:33, 17.02s/it]Score: 524
Episode:  20%|██        | 2/10 [00:33<02:13, 16.64s/it]Score: 611
Episode:  30%|███       | 3/10 [00:49<01:54, 16.38s/it]Score: 892
Episode:  40%|████      | 4/10 [01:05<01:38, 16.42s/it]Score: 330
Episode:  50%|█████     | 5/10 [01:24<01:25, 17.05s/it]Score: 665
Episode:  60%|██████    | 6/10 [01:40<01:07, 16.84s/it]Score: 588
Episode:  70%|███████   | 7/10 [01:56<00:49, 16.50s/it]Score: 522
Episode:  80%|████████  | 8/10 [02:12<00:32, 16.39s/it]Score: 768
Episode:  90%|█████████ | 9/10 [02:28<00:16, 16.25s/it]Score: 840
Episode: 100%|██████████| 10/10 [02:46<00:00, 16.62s/it]Score: 875
Mean Score: 662.09 (Std: 172.77)
```

DAGGER

Now we can implement DAgger, you have downloaded a relatively well trained model you can use as an expert for this purpose.

Load expert model that is provided as ONNX file.

Load the expert

```
# Load expert
expert_model = ConvertModel(onnx.load("expert.onnx"))
expert_model = expert_model.to(device)
# Freeze expert weights
for p in expert_model.parameters():
```

```
        p.requires_grad = False

expert_policy = Agent(expert_model, device)
```

```
/usr/local/lib/python3.11/dist-packages/onnx2pytorch/convert/layer.py:30: UserWarning: The given NumPy array is not writable, and Py
  layer.weight.data = torch.from_numpy(numpy_helper.to_array(weight))
```

```
#check expert performance

scores = []
n_eval_episodes = 10

for i in range(n_eval_episodes):
        score = run_episode(expert_policy, show_progress=False, capture_video=False)
        scores.append(score)
        print(f"Episode {i}, Score={score:.2f}")

mean_score = np.mean(scores)
std_score  = np.std(scores)
print(f"\nExpert Average Score over {n_eval_episodes} episodes: {mean_score:.2f} ± {std_score:.2f}")
```

```
Episode 0, Score=789.73
Episode 1, Score=710.90
Episode 2, Score=346.31
Episode 3, Score=648.41
Episode 4, Score=599.66
Episode 5, Score=917.60
Episode 6, Score=629.90
Episode 7, Score=414.53
Episode 8, Score=879.02
Episode 9, Score=638.98

Expert Average Score over 10 episodes: 657.50 ± 172.47
```

Next, you have to implement the DAgger algorithm (see slides for details). This function implements the core idea of DAgger:

1. Choose the policy with probability beta
2. Sample T-step trajectories using this policy
3. Label the gathered states with the expert

The aggregation and training part are already implemented.

```
# inner loop of DAgger
def dagger(env, train_policy, expert_policy, dataset, beta=1.,max_steps=4000):

        ######################
        ### YOUR CODE HERE ###
        ######################

        # Implement DAgger algorithm here
        # 1) Choose a policy (sample according to beta)
        # 2) Sample T-step trajectory with the chosen policy
        #     (T can be an entire episode or a single state, think about what makes more sense here and implement it
        # 3) Label the state (or states) with your expert if they come from your training policy

        #### Note ####
        # To get an action for the current state from your training policy or expert policy:
        # action = policy.select_action(state)
        #
        # Your training policy requires a single grayscale state while
        # the expert policy requires four stacked grayscale states
        # You can prepare your state for the policy like so:
        # Train policy:
        #          np.expand_dims(state[-1], 0)
        # Expert policy:
        #          state


        # Due to the RecordState wrapper you can get the states from the environment by calling
        # env.render()
        # Doing so will clear the list and the next time you call .render() will return the new states since the last
        # Note: be careful with the last state

        # Finally, add collected states and the actions the expert would execute in them to the dataset
```

```
    # dataset.append(states, actions)

    """
    在env上跑一段episode，使用混合策略 (beta概率选专家动作，(1-beta)概率选训练策略)，
    同时用专家策略对访问到的state打标签并追加到dataset中。
    """
    states_buffer = []
    actions_buffer = []

    obs, info = env.reset()
    done = False
    steps = 0

    while (not done) and (steps < max_steps):
        steps += 1

        # 1) 混合策略：随机从专家和训练好的策略中采样动作
        if np.random.rand() < beta:
            action_exec = expert_policy.select_action(obs)
        else:
            action_exec = train_policy.select_action(obs)

        # 与环境交互
        next_obs, reward, terminated, truncated, info = env.step(action_exec)
        done = terminated or truncated

        # 2) 始终用专家来给当前obs(或 next_obs)打标签
        #    这里假设我们要为 “我们所处的这个时刻” 存储(state, expert_action)。
        #    你也可以改为对 next_obs 做标注。
        expert_action = expert_policy.select_action(obs)

        # 需要转成 numpy 格式，以便 dataset.append() 保存
        if torch.is_tensor(obs):
            obs_to_store = obs.cpu().numpy()
        else:
            obs_to_store = obs

        # 记录
        states_buffer.append(obs_to_store)  # shape(4,84,84)
        actions_buffer.append(np.array(expert_action, dtype=np.int32))

        obs = next_obs

    # 将收集到的数据添加到 dataset
    dataset.append(states_buffer, actions_buffer)
```

Put everything together now.

1. Create new samples using the DAgger algorithm
2. Continue training your agent
3. Export your fully trained agent as an ONNX file

```
# Specify the google drive mount here if you want to store logs and weights there (and set it up earlier)
logger = Logger("logdir_dagger")
print("Saving state to {}".format(logger.basepath))

# start environment
env = make_env(seed=42, capture_video=False)

# Training
######################
### YOUR CODE HERE ###
######################
import torch.optim as optim
optimizer_dagger = optim.Adam(model.parameters(), lr=1e-4, weight_decay=1e-5)

n_bc_epochs = 2
for epoch in range(n_bc_epochs):
    train_loss, train_ent = train_one_epoch(model, train_loader, optimizer_dagger)
    val_loss, val_acc = validate(model, val_loader)
    print(f"[BC Epoch {epoch+1}/{n_bc_epochs}] train_loss={train_loss:.4f}, val_loss={val_loss:.4f}, val_acc={val_acc:.4f}")
```

```
# 迭代若干次 DAgger
n_dagger_iters = 5          # 例如迭代3轮
n_dagger_train_epochs = 2   # 每次收集数据后再训练多少epoch
beta = 1.0                  # 初始专家概率

for i in range(n_dagger_iters):
    print(f"\n=== DAgger Iteration {i+1}/{n_dagger_iters} (beta={beta:.2f}) ===")

    # 1) 采样一条(或多条)轨迹，用expert打标签并存进 dataset
    dagger(env, train_policy, expert_policy, train_set, beta=beta)

    # 2) 用新的训练集训练：此时 train_set 增加了新的 (state,action)
    #    重新构造 DataLoader 让其包含最新的数据
    dagger_train_loader = torch.utils.data.DataLoader(
        train_set, batch_size=64, shuffle=True, drop_last=False, pin_memory=True
    )
    for e in range(n_dagger_train_epochs):
        tr_loss, tr_ent = train_one_epoch(model, dagger_train_loader, optimizer_dagger)
        val_loss, val_acc = validate(model, val_loader)
        print(f"   [Epoch {e+1}/{n_dagger_train_epochs}] train_loss={tr_loss:.4f}, val_loss={val_loss:.4f}, val_acc={val_ac

    # 3) 逐步减少 beta，让代理更多地用自己的动作执行 (可选)
    #    常见做法是 beta = beta * decay_rate 或 beta = beta - 1/n_iters
    beta *= 0.9  # 这里只是示例写法

    # 4) 记得更新 train_policy(内部的 model 已更新梯度)
    train_policy = Agent(model, device)


save_as_onnx(model, sample_state, logger.onnx_file)
env.close()
```

```
Saving state to logdir_dagger/2025-03-25T21-15-27
[BC Epoch 1/2] train_loss=0.7602, val_loss=0.7776, val_acc=0.7066
[BC Epoch 2/2] train_loss=0.7588, val_loss=0.7806, val_acc=0.7094

=== DAgger Iteration 1/5 (beta=1.00) ===
   [Epoch 1/2] train_loss=0.7579, val_loss=0.7805, val_acc=0.7058
   [Epoch 2/2] train_loss=0.7558, val_loss=0.7774, val_acc=0.7075

=== DAgger Iteration 2/5 (beta=0.90) ===
   [Epoch 1/2] train_loss=0.7554, val_loss=0.7843, val_acc=0.7069
   [Epoch 2/2] train_loss=0.7544, val_loss=0.7827, val_acc=0.7046

=== DAgger Iteration 3/5 (beta=0.81) ===
   [Epoch 1/2] train_loss=0.7534, val_loss=0.7785, val_acc=0.7082
   [Epoch 2/2] train_loss=0.7514, val_loss=0.7852, val_acc=0.7075

=== DAgger Iteration 4/5 (beta=0.73) ===
   [Epoch 1/2] train_loss=0.7521, val_loss=0.7832, val_acc=0.7060
   [Epoch 2/2] train_loss=0.7497, val_loss=0.7851, val_acc=0.7041

=== DAgger Iteration 5/5 (beta=0.66) ===
   [Epoch 1/2] train_loss=0.7490, val_loss=0.7855, val_acc=0.7083
   [Epoch 2/2] train_loss=0.7451, val_loss=0.7828, val_acc=0.7090
```

```
n_eval_episodes = 10
```